Hardware Design

**Lab 1 Report**

Gate-Level Verilog

**Team 01**

112062144 謝佳晉 112062144 范升維

Table of Contents:

[Q1. 1x4\_4bit DMUX 1](#_Toc177556472)

[A. 1x2\_4bit DMUX 1](#_Toc177556473)

[B. 1x4\_4bit DMUX 1](#_Toc177556474)

[Q2. 2x2\_4bit Crossbar Switch 2](#_Toc177556475)

[Q3. 4x4\_4bit Crossbar Switch 3](#_Toc177556476)

[Q4. Toggle Flip-Flop 4](#_Toc177556477)

[Testbenches 5](#_Toc177556478)

[A. 1x4\_4bit DMUX 5](#_Toc177556479)

[B. 2x2\_4bit Crossbar Switch 5](#_Toc177556480)

[C. 4x4\_4bit Crossbar Switch 6](#_Toc177556481)

[D. Toggle Flip Flop 7](#_Toc177556482)

[What we have learned from Lab1? 7](#_Toc177556483)

[Contributions 8](#_Toc177556484)

# Q1. 1x4\_4bit DMUX

## A. 1x2\_4bit DMUX

* 1 input: *in[3:0]*
* 1 control signal:  *sel*
* 2 outputs: *a[3:0], b[3:0]*

▲ Figure 1.1

First, we construct module 1x2 DMUX. This module can **route the input**, *in[3:0]*, to either a or b **depending** on the **control signal**, *sel*. When *sel* is 0, the upper four AND gates have one of the input 1, so their outputs will be the same as input. Since, the lower four have one of the input 0, their outputs will all be 0. Vice versa.

## B. 1x4\_4bit DMUX

* 1 input: *in[3:0]*
* 1 control signal: *sel[1:0]*
* 4 outputs: *a[3:0], b[3:0], c[3:0], d[3:0]*

▲ Figure 1.2

Now we have 1x2 DMUX, so we can construct module 1x4 DMUX by **combining three** 1x2 DMUX. We can use the left 1x2 DMUX with *sel[1]* as its control signal to preliminarily route the input to *a*/*b* or *c*/*d*. Next, we use the right two 1x2 DMUX with *sel[0]* as its control signal to route the input to *a* or *b*, and *c* or *d*. Therefore, we get one of *a*/*b*/*c*/*d* to be the same as input, and other three outputs be 4’b0000.

# Q2. 2x2\_4bit Crossbar Switch

▲ Figure 2.1: 2x1\_4bit MUX ▲ Figure 2.2: 2x2\_4bit Crossbar Switch

* 2 inputs: *in1[3:0], in2[3:0]*
* 1 control signal: *control*
* 2 outputs: *out1[3:0], out2[3:0]*

We use two 1x2 DMUX and two 2x1 MUX to construct a 2x2 Crossbar Switch.

When *control* is 0, *in1* → *con1* → *out1*, and *in2* → *con4* → *out2*, like parallel.

When control is 1, *in1* → *con2* → *out2*, and *in2* → *con3* → *out1*, like cross.

# Q3. 4x4\_4bit Crossbar Switch

▲ Figure 3

* 4 inputs: *in1[3:0], in2[3:0], in3[3:0], in4[3:0]*
* 1 control signal: *control[4:0]*
* 4 outputs: *out1[3:0], out2[3:0], out3[3:0], out4[3:0]*

The 4x4 Crossbar Switch consists of five 2x2 Crossbar Switch. It acts like an enhanced version of 2x2 Crossbar Switch, but there has a limit. Some of the inputs are impossible to route to some specific outputs. The following are the routes that **cannot be implemented** by this 4x4 Crossbar Switch:

[(*in1*, *out3*), (*in2*, *out4*), (*in3*, *out1*), (*in4*, *out2*)],

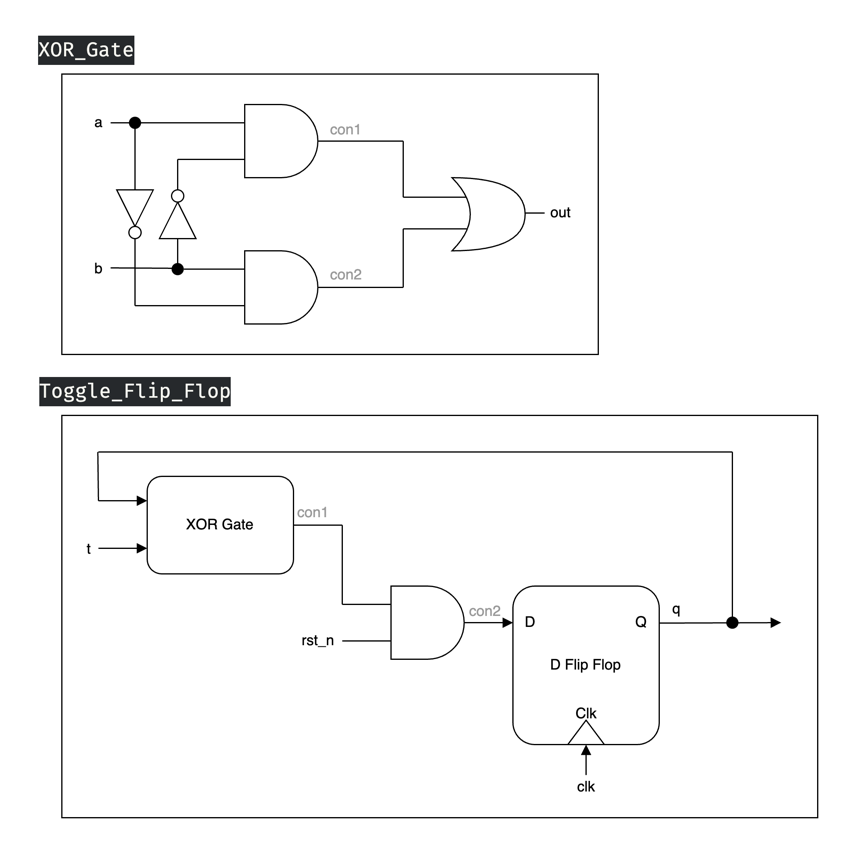
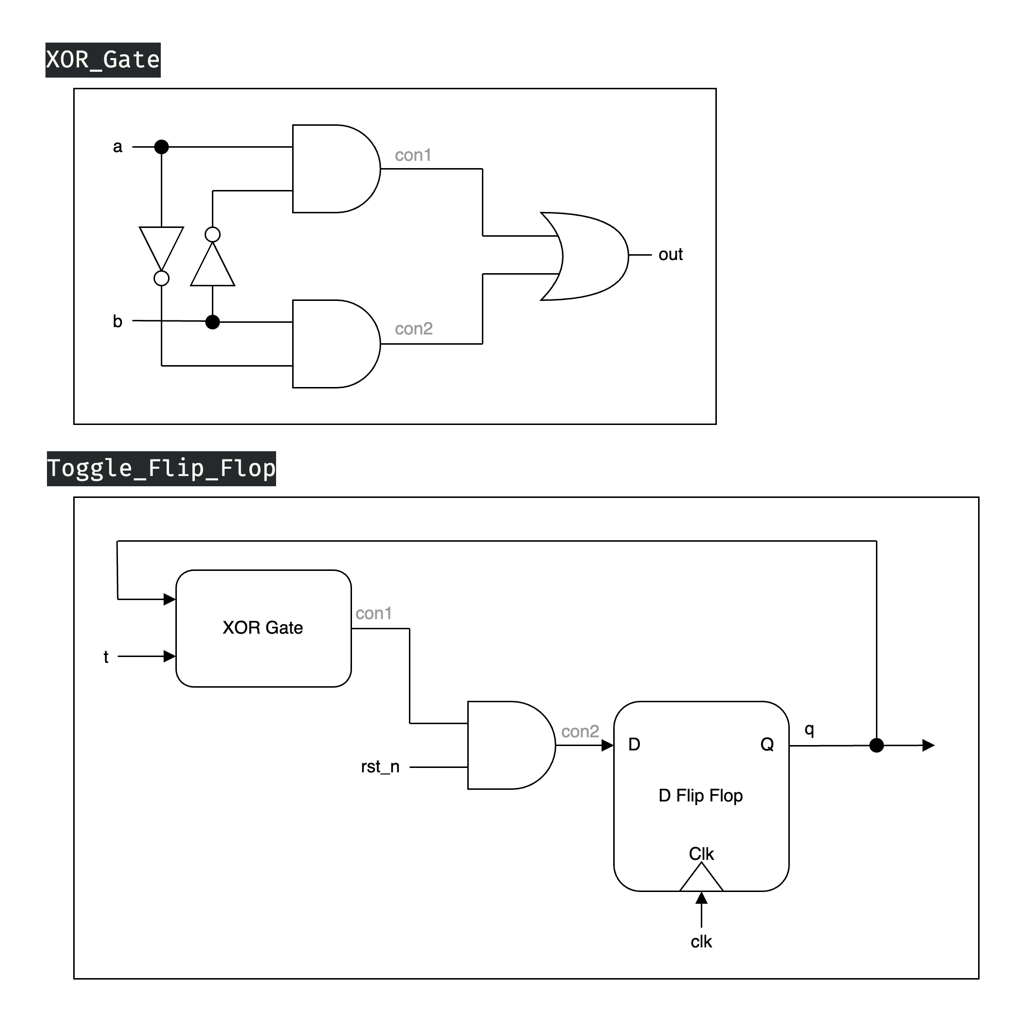
[(*in1*, *out4*), (*in2*, *out3*), (*in3*, *out1*), (*in4*, *out2*)],

[(*in1*, *out3*), (*in2*, *out4*), (*in3*, *out2*), (*in4*, *out1*)],

[(*in1*, *out4*), (*in2*, *out3*), (*in3*, *out2*), (*in4*, *out1*)].

The above limits exist because **only** **one** of the *in1*/*in2* **can be routed** to *out3*/*out4*, and at the same time, only one of the *in3*/*in4* can be routed to *out1*/*out2*. To handle the problem, we can **add an additional 2x2 Crossbar** which takes *con1* and *con4* as input.

# Q4. Toggle Flip-Flop

 ▲ Figure 4.1: D-Latch ▲ Figure 4.2: D Flip-Flop

▲ Figure 4.3: XOR Gate ▲ Figure 4.4: Toggle Flip-Flop

* 3 inputs: *clk, t, rst\_n*
* 1 output: *q*

Given that we cannot utilize built-in XOR gate, Figure 4.1 shows how we construct the XOR gate ourselves. After we have the XOR gate and DFF, from Lab1 basic question 2, we can further get a Toggle Flip Flop as shown in Figure 4.2.

When *rst\_n* is 0, and on the **positive edge**, *q* will be reset to 0.

When *rst\_n* is 1, and on the positive edge:

When *t* is **stably 1** at that time, *q* will be toggled into *~q.*

If *t* is 0 or *t* isn't **stable** at its **setup time** and **hold time**, *q* will not be changed.

# Testbenches

## 一張含有 螢幕擷取畫面, 文字, 行 的圖片 自動產生的描述A. 1x4\_4bit DMUX

We can test the module by adding up *sel'*s value and inspect whether the outputs change correspondingly, with ever-changing input value to demonstrate its flexibility.

## B. 2x2\_4bit Crossbar Switch

**一張含有 螢幕擷取畫面, 鍵盤, 電腦 的圖片

自動產生的描述**

We can test this module by repeatedly changing the value of control to see if the inputs are routed to the correct output ports. Similarly, we keep changing the input values to demonstrate its flexibility.

## 一張含有 螢幕擷取畫面, 文字 的圖片 自動產生的描述C. 4x4\_4bit Crossbar Switch

We can test this module by changing control’s value to see if the inputs have been routed to the correct output ports, just like 2x2\_4bit Crossbar Switch, along with changing input value to demonstrate its flexibility. **To find the impossible routes easily**, we can set input values to be a, b, c, d and use **$monitor** command.

**◎ Difference between $display and $monitor:**

**$display**: It's like printf in C. It output the message once the statement is executed.

**$monitor**: Once being set, every time the signal changed, it will output the message.

## D. Toggle Flip Flop

We can test this module by setting the proper staggered interval(4, 37, 10) so that we can simulate more cases that it may meet.

# What we have learned from Lab1?

We began writing Verilog for the first time, as last semester's logic design course only involved "looking" at it, without hands-on writing experience. In this lab, we reviewed some previous concepts such as latches and flip-flops, and learned how to implement them using Verilog. We also discovered that using gate-level descriptions to write buses requires manually typing many lines of code. Additionally, we learned how to write testbenches and use Vivado simulation to verify if our modules were functioning correctly. Moreover, we used draw.io for the first time, which took some time to become familiar with in order to create neat, concise, and visually appealing circuit diagrams. Lastly, we spent considerable time learning how to create a more formal report with a table of contents. In conclusion, we learned a great deal of content through this lab experience.

# Contributions

謝佳晉：

wrote Q1~Q4 Verilog

wrote testbench Q1~Q4

performed Q1~Q4 simulation on Vivado

drew diagram Q3, Q4

constructed report foundation with images and descriptions

made FPGA demonstration

范升維：

wrote Q1~Q4 Verilog

wrote testbench Q1~Q4

performed Q1~Q4 simulation on Vivado

drew diagram Q1, Q2

revised diagram Q3, Q4

organized, beautified report, added more descriptions and revised images

implemented FPGA demonstration